On generating sentinel pointer values in Windows

devblogs.microsoft.com/oldnewthing/20170420-00

April 20, 2017

Raymond Chen

Suppose you have a need for sentinel pointer values. Let's say that your function operates on pointers to `Widget` objects, but you need a few special values that convey special meaning, like "There is no widget" or "Please use a default widget" or "Inherit the widget from the parent object" or "All the widgets."

Well, many languages give you a sentinel pointer value up front, typically called `null` or `Nothing` or `nullptr` or something emptyish like that. If you need only one sentinel value, then that's a pretty simple choice.

On the other hand, this emptyish sentinel value is a common thing that could be generat3ed by mistake. Some languages use it as the default value for pointers. And the emptyish value might come out of an earlier failed operation, like an allocation. So you might want to avoid using the emptyish value as a sentinel because it is too easy to pass by mistake.

If you need a small number of sentinel values, you could just allocate a few objects for the sole purpose of providing an address. Some classes in the C++ standard library do this. For example, `std::map` might allocate a sentinel value to represent `end()`. (That sentinel value serves other purposes, too.)

Another idea is to create a bunch of addresses for your own use and carve your sentinel values out of them. You can `VirtualAlloc(MEM_RESERVE)` some address space, and nothing will go into that address space unless you put it there. If you reserve the address space and intentionally put nothing in it, then all the addresses in that reserved region are potentially usable as sentinels.

Windows itself does this for you: As part of setting up the process address space, the kernel reserves the bottom 64KB of address space, so no valid objects will be allocated there. That gives you 65536 sentinel values, although one of them matches `nullptr`, so it's 65535 *new* sentinel values. This is the technique used by the `MAKEINTRESOURCE` and `MAKEINTATOM` macros to allow an integer to be smuggled inside a string pointer.

Prior to Windows 8, applications could unreserve the bottom 64KB of address space and allocate actual memory there, which created the opportunity for mass confusion. Windows 8 put a stop to that.

If your widget object has alignment requirements (and if it consists of anything other than raw bytes, it probably does), you can use any pointer value that does not conform to those requirements. For example, if widgets must be 4-byte aligned, then any pointer value which is not divisible by four can be used as a sentinel, since it will never match the address of a valid widget.

If your widget object has no alignment requirements, you could always invent one by using a declaration appropriate to your toolset, such as `__declspec(align(2))` or `__attribute__(aligned(2))` or whatever.

Even if your alignment requirements are only word-alignment, that gives you two billion possible 32-bit sentinel values, which is quite a lot. You can use the encoding $f(n) = n \times 2 + 1$ to create a sentinel and its inverse $g(n) = (n - 1) / 2$ to convert a sentinel back to its magic number.

(And if you're using 64-bit pointers, then the number of possible sentinel values is staggering.)

Exercise: Critique the following suggestion: "You can pick any value greater than `0x80000000` to use as a sentinel value."

Raymond Chen

Follow